

1

Avec la cassette de vidéo-jeux et programmes pour ZX SPECTRUM et VIC 20

Le pseudocode et la programmation de base

Les touches fonctionnelles et le joystick du Vic 20

Comment augmenter la vitesse des programmes en Basic

La mémorisation des données du ZX Spectrum

Sinclair & Commodore: les nouveautés



Promopublications



A l'ère de l'informatique SOFTHEQUE est heureuse de vous présenter la première revue qui peut être utilisée sur ordinateur.
Chaque mois nos lecteurs trouveront dans cette publication les indications indispensables pour apprendre à mieux connaître et utiliser leur micro-ordinateur. Ils trouveront aussi avec chaque recueil, une précieuse cassette de logiciel.
Nous espérons que les numéros de la revue et les cassettes qui les accompagnent trouveront leur place à côté des micro-ordinateurs dont ils se proposent de vous révéler les moindres secrets.
Allier la théorie à la pratique, la règle s'impose plus que jamais en informatique. Nous sommes convaincus que les jeux vidéo peuvent avoir une excellente fonction formatrice à condition que le joueur comprenne bien la logique du programme de jeu qu'il introduit dans son ordinateur.
SOFTHEQUE vous propose donc une super collection de recueils d'informations et de logiciel spécialement conçue pour les utilisateurs des micro-ordinateurs ZX Spectrum et Commodore VIC 20.
Nos lecteurs seront donc tous branchés sur la même longueur d'onde et nous espérons qu'ils formeront une grande famille qui nous aidera à mieux travailler et répondre à leurs exigences.
Nous vous donnons donc rendez-vous au mois prochain et vous rappelons que nous sommes à votre entière disposition.

LE PSEUDOCODE ET LA PROGRAMMATION DE BASE

Deux sortes de lecteurs s'intéresseront à ces pages: l'un est le programmeur BASIC, l'autre le nouvel enthousiaste de la programmation BASIC. Mais commençons par le commencement...

Il existe un langage efficace, appelé "pseudocode", grâce auquel il est possible d'exprimer, de manière non ambiguë, élégante et précise, n'importe quel programme, pensé par n'importe quel programmeur, et réalisable en n'importe quel langage de programmation existant (BASIC, COBOL, FORTRAN, PASCAL, PL/1 etc.). Le "pseudocode" ("pseudocode" en anglais!) a été mis au point au cours de ces vingt dernières années par les efforts communs des programmeurs du monde entier.

Cela signifie qu'il est garanti par l'expérience de vingt années de pratique de programmation de la communauté spécialisée dans l'écriture de programmes professionnels.

LE PSEUDOCODE

La faculté d'exprimer ses propres programmes au moyen du "pseudocode" est une qualité recherchée car elle permet de démontrer sa propre compétence concernant l'écriture de programmes dans le monde du travail. Il arrive souvent que des programmeurs possédant des années d'expérience soient invités à suivre des cours de "pseudocode" pour renouveler et enrichir leur bagage culturel spécifique.

Mais en quoi cela peut-il intéresser ceux qui ne savent pas écrire de programmes?

C'est très simple: le pseudocode se révèle comme le meilleur instrument permettant d'apprendre l'art de la programmation.

Il est tout particulièrement indiqué pour les jeunes programmeurs, ou aspirants programmeurs, qui veulent avoir tous les atouts en main pour devenir des programmeurs de premier ordre.

Dans le monde du Personal Computer (ordinateur personnel), le "langage franc" d'échange culturel sera sans doute le BASIC, pour les dix années à venir.

Le BASIC appartient cependant à une génération de langage datant de deux générations au moins, et force nous est de reconnaître qu'il ne peut suivre



l'évolution parallèle du pseudocode, qui a conduit à projeter les langages modernes, tels que le Pascal. En bref, avec le Basic et seulement avec le Basic, on court le risque de ne jamais posséder cette capacité d'écrire les programmes élégants et efficaces qui, dans le monde de la programmation professionnelle, sont désormais considérés comme standard.
Pour le programmeur amateur, la perspective est encore plus alléchante: comprendre le "pseudocode" signifie comprendre la Programmation, avec un "P" majuscule, une fois pour toutes indépendamment du langage Basic, et acquérir automatiquement la technique moderne correcte de l'écriture des programmes.
Les pages qui suivent vous apportent la base nécessaire. Vous apprendrez ainsi à écrire et à projeter des programmes avec le pseudocode pour passer ensuite au langage BASIC.

LE LANGAGE

L'élégance et la substance d'un procédé algorithmique, c'est-à-dire d'un programme, sont davantage mises en évidence si on utilise un jargon de spécialiste plutôt que le langage normal. Cela se produit dans n'importe quel domaine professionnel: certains termes linguistiques acquièrent une signification bien précise, et certaines constructions syntaxiques deviennent obligatoires.
Prenons un exemple: dans le domaine de l'électronique ou de la Hi-Fi, dire "amplification" sous-entend amplification électronique, alors que dans le domaine de l'architecture, cela peut signifier l'amplification mécanique de certains locaux; dans le jargon informatique, il est normal de dire qu'un programme "tourne" pour dire qu'il fonctionne ou est en cours d'exécution; essayez de le dire à quelqu'un qui ne fait pas partie de ce secteur et vous vous rendrez compte à quel point cette tournure de phrase peut sembler incorrecte et désagréable.
Pour construire un jargon, c'est-à-dire un langage spécialisé, nous devons établir des REGLES de construction des phrases de ce langage, dans lesquelles nous spécifions les PAROLES avec un signifiant conventionnel et les CONSTRUCTIONS linguistiques qui sont permises.
Le pseudocode, qui est un langage très puissant, a une règle précise concernant les paroles que l'on peut utiliser.

R.1 Tous les verbes de la langue sont permis, à condition qu'ils ne soient pas ambigus.

Une phrase permise en pseudocode peut donc être la suivante:

CALCULER la moyenne entre les 10 chiffres susdits:

Le verbe "CALCULER" est permis, parce qu'il exprime une action non ambiguë.
En ce qui concerne les constructions syntaxiques, c'est-à-dire la manière dont les phrases verbales s'articulent entre elles, le pseudocode a une seconde série de règles.

R.2 La succession de phrases verbales simples est permise; ces phrases doivent être écrites à la suite l'une de l'autre.

Exemple: La succession suivante est permise:

Ajouter un oeuf dans la poêle.
Mettre une pincée de sel.

(Il ne faut pas s'étonner du fait que nous exprimons une recette de cuisine en pseudocode: nous vous avons dit que le pseudocode est très puissant!)

R.3 On peut construire une ALTERNATIVE de phrases, à condition qu'il y ait un prédicat de contrôle.

On entend par prédicat de contrôle une phrase qui énonce une condition. Nous le verrons mieux par la suite; pour le moment, étudions l'exemple:
Exemple - La période suivante est exprimée en pseudocode:

S'il y a trop peu de beurre
ALORS ajoutez de l'huile
AUTREMENT mettez un peu de beurre
FIN_SI

Comme vous avez pu voir (si vous n'avez pas encore faim), les recettes de cuisine sont le meilleur exemple du contexte linguistique dans lequel il est nécessaire de spécialiser le langage, et dans lequel la structure algorithmique de la tâche à remplir est la plus évidente. Effectivement, un ALGORITHME est un procédé qui peut être communiqué grâce à une série d'instructions à suivre dans l'ordre et selon les modalités indiquées.
Une recette de cuisine est un ALGORITHME qui permet, à tous ceux qui savent lire le français et sont susceptibles de suivre les constructions syntaxiques du texte, de réussir le bon petit plat prévu.
Un PROGRAMME n'est autre que la réalisation d'un ALGORITHME dans un langage de programmation, de manière à ce qu'une machine produise le résultat prévu. La personne qui lit une recette et obtient le bon petit plat prévu réalise un programme, écrit par le chef qui a préparé la recette.
Le pseudocode est capable de formuler (donc d'écrire sous forme standard) n'importe quel procédé algorithmique.
Cette affirmation est également vraie dans un sens plus profond qu'il n'y apparaît: il existe un théorème mathématique qui affirme qu'un procédé de n'importe quel genre, même complexe ou difficile à expliquer, peut être FORMULE comme un ALGORITHME.



Cependant, les deux constructions que nous avons indiquées jusqu'ici, c'est-à-dire la SERIE et l'ALTERNATIVE, ne suffisent pas à tout exprimer. On pourrait démontrer, au contraire, qu'avec ces deux constructions uniquement, les algorithmes que nous serions capables d'exprimer seraient les plus simples et les moins "intelligents". Effectivement ce qui manque est précisement la capacité d'exprimer un processus qui puisse SE REGLER DE LUI-MÊME. Prenons un exemple. Si, avec une recette de cuisine, on voulait expliquer comment se fait un oeuf battu, on pourrait écrire, avec la capacité linguistique acquise jusqu'ici:

RECETTE DE L'OEUF BATTU
Version 1
SI vous n'avez pas très faim
ALORS mettez un oeuf dans le bol.
AUTREMENT mettez deux oeufs dans le bol.
FIN_SI
Ajoutez trois
cuillerées de sucre
Battez l'oeuf pendant 10 minutes.

Nous avons utilisé uniquement des âlternatives et des séries de phrases. Eh bien, cet algorithme est stupide, parce qu'il décide de battre l'oeuf pendant 10 minutes, indépendamment du résultat obtenu, qui peut être bon certaines fois et d'autres non. Si notre batteur électrique s'arrête au contraire juste quand l'oeuf a pris cette couleur blanc-doré qui prouve qu'il est correctement battu, ne serions-nous pas prêts à attribuer au batteur électrique une intelligence supérieure? Sans aucun doute, et ceci est également ce que nous confirme la cybernétique, ou science de l'intelligence artificielle et des mécanismes de communication:

— Le signe d'un comportement intelligent est la capacité d'influer sur la situation de l'environnement jusqu'à l'obtention d'une configuration optimale.

C'est pour cela que, dans nos ordinateurs, a été insérée également la capacité de produire une manière de modifier une situation extérieure jusqu'à ce que l'on obtienne la situation optimale. Et, en pseudocode, nous aurons donc la construction REPETITION qui s'utilise de la manière suivante:

Exemple:

RECETTE DE L'OEUF BATTU
Version 2
SI vous n'avez pas très faim
mettez ALORS un oeuf dans le bol
AUTREMENT mettez deux oeufs dans le bol.
FIN_SI
CONTINUEZ JUSQU'À ce que vous obteniez une couleur blanc-doré.
Battez l'oeuf
FIN_CONTINUEZ

La dernière phrase met en évidence le fait que l'action "battez l'oeuf" est répétée jusqu'à ce que le mélange obtienne la couleur jaune doré.

R.4 La construction REPETITION d'un certain nombre de phrases est permise et la répétition de ces phrases est conditionnée par un prédicat de contrôle.

PROCÉDÉS ALGORITHMIQUES AVEC L'ORDINATEUR PERSONNEL.

Maintenant que nous avons indiqué les règles de construction d'un algorithme en pseudocode, nous devons concentrer notre attention sur l'ensemble d'objets et d'actions qui interviennent dans les algorithmes qui sont réalisés par des programmes BASIC.

Ce monde d'objets est constitué d'éléments situés à deux niveaux d'abstraction seulement

- — noms de variables
- — valeurs explicites.

Supposons que j'écrive l'algorithme suivant:

additionnez 3 et 2.

Ceci est sans aucun doute un minuscule algorithme constitué par la séquence d'une seule phrase.

Dans cet algorithme, je fais explicitement référence aux valeurs 3 et 2. Cet algorithme effectue la somme de 3 et 2 et produit le résultat 5.

Ecrivons un autre algorithme:

additionnez A et B

Dans cette phrase, A et B sont les noms d'objets qui contiennent des valeurs numériques et le sens de la phrase est: regardez quelle valeur a la variable A et additionnez-la à la valeur qu'a la variable B.

Dans cet exemple, A et B sont des NOMS DE VARIABLES.

On ne répétera jamais assez que, dans notre propre langue, nous sommes très capables de comprendre immédiatement quand nous sommes en train d'utiliser un nom de variable et quand nous utilisons une valeur explicite.

Cette qualité que nous possédons est tellement automatique et spontanée qu'il est difficile de faire la différence entre les deux choses, le nom et la valeur explicite, et ceci représente une difficulté pour tous les néophytes de la programmation.

Par exemple, si nous entendons quelqu'un dire à un ami: "Additionne 5 et le chiffre auquel tu as pensé", nous ne nous rendons pas compte que, dans sa phrase, il y a: une valeur explicite (5) et un nom de variable: "le chiffre auquel tu as pensé". Seul l'ami, qui connaît la valeur du chiffre auquel il a pensé, est capable d'effectuer l'algorithme. Mais le personnage qui prononce la phrase est capable de construire un algorithme même sans connaître les valeurs ex-

plicites impliquées dans les calculs, en nommant simplement des variables.
Le constructeur d'algorithme fait le même raisonnement: chaque fois qu'il veut nommer des objets dont les valeurs seront connues de la machine en cours d'exécution, il utilise des noms de variables.
Nous arrivons ainsi à la Règle R5.

R5. Il est permis de nommer des variables en choisissant des noms quelconques et de les citer comme objets de nos phrases, en prenant uniquement la précaution de faire correspondre, à un même nom, une même variable.

En écrivant les programmes, ce n'est que rarement que nous voudrions que l'ordinateur invente la valeur initiale d'une variable, comme dans le cas de l'exemple où nous demandions à notre ami de penser à un chiffre au hasard. Dans les programmes, nous ferons en général une opération **d'attribution** d'une valeur explicite aux variables que nous voudrons affecter d'une valeur initiale.
L'opération d'attribution s'écrit de diverses manières en pseudocode, mais la plus courante est la suivante:

A = 3

Ce qui signifie: la variable A contient la valeur 3.
Dans un programme, une succession d'assignations modifie la valeur de la variable A.
Par exemple:

A = 4
B = 5
A = 3
:
:

Actuellement, A vaut 3 et B vaut 5.
La phrase:

C = A + B

signifie donc: ajoute la valeur **actuelle** de A à la valeur **actuelle** de B et attribue la valeur de la somme ainsi obtenue à la variable C.
Donc, l'algorithme:

A = 5
B = 9
C = (A + B) / 2 $\left(= \frac{A+B}{2}\right)$
Ecrire C

Ecrire la valeur de C à la fin de l'opération: (5 + 9)/2 c'est-à-dire: 7.
Par convention, on établit que les calculs à droite d'égal sont effectués en premier et donc, si la même variable est citée à droite et à gauche du signe égal, la nouvelle valeur ne sera attribuée à la variable citée qu'à la fin de l'opération:

A = 5
B = 9
B = A + B
Ecrire B

C.à.d. 14, et l'instruction B = A + B se lit ainsi:
additionnez la valeur actuelle

de B (c.a.d. 9) et la valeur actuelle de A (c.a.d. 5) et mettez le résultat (c.a.d. 14) en B.

Nous sommes déjà capables d'exprimer un algorithme sous forme de pseudocode. Donnons un algorithme simple: produire la somme des 10 premiers nombres naturels (c.a.d. 1+2+3+4+5+6+7+8+9+10).

En pseudocode, nous pouvons écrire:

Programme pour obtenir la somme des 10 premiers nombres:

```
1 NUM = ∅ (Pour
  différencier le chiffre zéro
  de la lettre O majuscule,
  votre ordinateur écrit zéro
  sous la forme ∅)
2 B = ∅
3 répétez jusqu'à NUM>1∅
4 B=B+NUM
5 NUM=NUM+1
6 Fin répétez
7 Ecrivez B
```

Comment l'algorithme fonctionne-t-il? Les lignes 1 et 2 attribuent la valeur initiale de 0 dans les variables NUM et B. L'instruction 3 dit de répéter les instructions 4 et 5 jusqu'à ce que NUM atteigne - et ne dépasse pas - le nombre 10.

A chaque passage, le programme ajoute à la variable B la valeur assumée par NUM, laquelle augmente d'une unité à chaque fois, en vertu de l'instruction 5.

En langage BASIC cet algorithme prend l'aspect suivant:

```
1∅ NUM=∅
2∅ B=∅
3∅ IF NUM>1∅ GOTO 7∅
4∅ B=B+NUM
5∅ NUM=NUM+1
6∅ GOTO 3∅
7∅ PRINT B
```

(Pour les initiés au langage BASIC, note: je sais moi aussi qu'en BASIC, il y a FOR' NEXT, etc., mais patientez un moment!...)

En pseudocode, le prédicat de contrôle est toujours une phrase de la forme a Rb, dans laquelle a et b sont des valeurs explicites ou des noms de variables, et R est un opérateur de RELATION, qui peut être >, <, ⩾, ⩽, =.

Exemple de prédicats:

A>15, B=5∅; A⩽2∅.

Rappelons que les prédicats de contrôle conditionnent les figures de l'alternative et de la répétition.

Les variables en pseudocode (et aussi en programmation) sont de deux types seulement: NUMERIQUES et ALPHA-NUMERIQUES.

Les variables numériques peuvent contenir des valeurs numériques (par exemple le chiffre 3 ou le nombre 15685); les variables alpha-numériques contiennent des mots écrits (par exemple "Bonjour, je m'appelle Marc"). Maintenant, rappelez-vous ce que nous disions à propos des noms de variables et des valeurs explicites. Quand nous écrivons 4, nous écrivons une valeur explicite, quand nous écrivons NUM, nous écrivons un nom de variable. Mais comment faire pour écrire une



valeur explicite alpha-numérique, sans la confondre avec le nom d'une variable?

La réponse est simple: nous l'écrivons entre guillemets, comme si nous faisions une citation:
La phrase:

A = "BASIC"

signifie en pseudocode deux

EXEMPLES

```
1∅ A = 5
2∅ B = 9
3∅ C = (A + B)/2
 7
```

```
A = 5
B = 9
C = (A + B)/2
PRINT C
```

Exemple 1 BASIC et Pseudocode:

```
1∅ NUM = ∅
2∅ B = ∅
3∅ IF NUM > 1∅ GOTO 7∅
4∅ B = B + NUM
5∅ NUM = NUM + 1
6∅ GOTO 3∅
7∅ REM fin répètez
55
```

```
NUM = ∅
B = ∅
Répètez jusqu'à NUM > 1∅
        B = B + NUM
        NUM = NUM + 1
FIN Répètez
Ecrivez B
```

Exemple 2: Somme des 10 premiers nombres entiers:

```
1∅ NUM = ∅
2∅ B = ∅
3∅ IF NUM > 1∅ GOTO 7∅
4∅ B = B + NUM
5∅ NUM = NUM + 1
6∅ GOTO 3∅
7∅ REM fin répètez
8∅ PRINT "la valeur de la somme des dix premiers nombres
   entiers est égale à" B.

RUN

La valeur de la somme des dix premiers nombres entiers est
égale à 55.
```

Exemple 3: Somme des dix premiers nombres.

choses:
1) que A est une variable alphanumérique (cela se déduit de son utilisation).
2) que A contient comme valeur initiale explicite le mot "BASIC", formé de 5 lettres.
La phrase: BASIC = "BASIC" n'est pas ambiguë:
BASIC est le nom d'une variable, tandis que "BASIC" est la valeur explicite que cette variable assume après que l'instruction ait été exécutée. Nous améliorons alors notre programme précédent en ajoutant un mot:

```
Num = 1
B = ∅
Répétez jusqu'à NUM > 1∅
B = B + NUM
NUM = NUM + 1
Fin_Répétez
```

Ecrire "La valeur de la somme des 10 premiers nombres est égale à", B.
La dernière phrase signifie que le mot entre guillemets, doit être écrit tel qu'il est indiqué, tandis que B (sans guillemets) signifie qu'il faut ensuite ajouter à la phrase la valeur numérique assumée par B à la fin du programme.
En BASIC, nous aurons:

```
1∅ NUM = 1
2∅ B = ∅
3∅ IF NUM > 1∅ GOTO 8∅
4∅ B = B + NUM
5∅ NUM = NUM + 1
6∅ GOTO 30
7∅ REM Fin Répétez
8∅ PRINT "La valeur de la
   somme des dix premiers
   nombres est égale à", B.
```

Le résultat du programme sera cette fois la ligne:
La valeur de la somme des 10 premiers nombres est égale à 55.
Notez que, en émettant le mot écrire (PRINT), le BASIC ne met pas les guillemets, parce que la valeur de la ligne est tout ce qui est contenu entre les guillemets.
C'est ici que se termine le premier article de la série. Les règles de la pseudocodification ne sont pas encore terminées. Les variables avec indice nous attendent encore, ainsi que les instructions spécialisées d'écriture, les lignes additionnelles et beaucoup d'autres choses encore.

Nous devons également clarifier la manière de TRADUIRE exactement toutes les figures de pseudocode en BASIC.
Pour ceci, patientez encore un peu, il vaut mieux pour l'instant les considérer comme déjà acquises au lieu de se comporter en bureaucrates et établir un certain nombre de règles de traduction. En attendant, vous pourrez vous exercer à comprendre la pseudocodification en jouant avec les trois mini-programmes en BASIC des exemples 1, 2 et 3. Etant donné que chacun de ceux-ci est la traduction exacte des algorithmes en pseudocode examinés jusqu'ici, vous pouvez déjà vous faire une idée et éventuellement anticiper vous-même ce que nous verrons ensemble prochainement.



DEUX NOUVEAUX CONCEPTS

Nous avons découvert le pseudocode. Nous avons vu les trois figures du pseudocode: série séquentielle, alternative et répétition. Nous avons parlé de variables et de valeurs explicites.
Cet article présente deux nouveaux concepts très importants, aussi bien en pseudocode qu'en BASIC: les variables avec indice, et les répétitions énumératives.
Avant d'aborder ces nouveaux points, il faut tout d'abord compléter l'aspect de l'écriture formelle des figures du pseudocode.

LA SÉRIE SÉQUENTIELLE

Pour ce qui concerne la SERIE, en supposant que les phrases se suivent l'une après l'autre, il n'existe pas de règles d'écriture, sauf celle qui consiste à les écrire en commençant à partir de **la même marge**; pour prendre un exemple, supposons que nous voulions écrire une série de N actions, Az1, Az2, ... Azn nous écrivons:

```
Action 1
Action 2
   :
   :
Action N
```

L'ALTERNATIVE

Prenons maintenant l'exemple de l'alternative.
Si les actions 1, 2 et 3 sont conditionnées par réponse VRAIE au prédicat de contrôle, et les actions 4, 5, 6 par la réponse FAUX au prédicat de contrôle, on écrit correctement:

```
SI prédicat
ALORS
  Act 1
  Act 2
  Act 3
AUTREMENT:
  Act 4
  Act 5
  Act 6
FIN_SI
```

Vous observerez que les actions subordonnées sont écrites plus à l'intérieur, à un certain nombre d'espaces par rapport à la position des paroles clées ALORS et AUTREMENT.
Ce décalage est FONDAMENTAL si l'on veut écrire un pseudocode clair et bien lisible: il faut EGALEMENT utiliser ce décalage dans le programme BASIC qui réalisera l'ALGORITHME.
La règle du décalage vaut aussi pour chacune des structures de contrôle suivantes susceptibles de prendre la place d'une Action de la liste; par exemple

```
SI prédicat 1
ALORS
  A 1
  SI prédicat
  ALORS
```

```
    Act 2
  AUTREMENT
    Act 3
  FIN_SI
AUTREMENT
  Act 4
  Act 5
  Act 6
FIN_SI
```

Dans cet exemple, on voit qu'une alternative successive implique un décalage à l'intérieur successif, qui s'annule quand FIN_SI termine l'alternative.
Souvent, dans la branche de AUTREMENT, on ne veut effectuer aucune action; dans ce cas on peut écrire:

```
SI prédicat
ALORS
  Act 1
  Act 2
AUTREMENT
FIN_SI
```

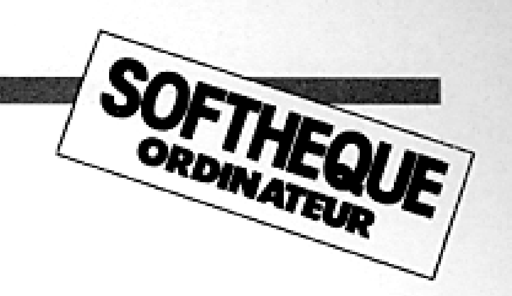
sans rien mettre dans la branche d'AUTREMENT.
Rappelons que **prédicat** est une phrase du type:
A>B, A<B, A>10, ou une combinaison quelconque de noms de variables, valeurs explicites et symboles de relation (c.à.d. >, <, =).
Passons maintenant à la répétition. Une répétition des actions A 1, A 2, A 3, conditionnée par le prédicat de contrôle P, s'écrit ainsi:

```
Répétez jusqu'à ce que P
  Act 1
  Act 2
  Act 3
FIN_RÉPÉTEZ
```

La règle FONDAMENTALE du décalage à l'intérieur vaut également dans le cas de la répétition: rappelons que c'est un aspect formel caractéristique du pseudocode et non un "optionel" que l'on peut éliminer.

LES VARIABLES AVEC INDICE

Résumons ce que nous avons vu jusqu'ici à propos des variables.
Les variables (vu la manière dont elles ont été présentées jusqu'ici) sont des noms qui sous-entendent des "BOITES" dans lesquelles sont emmagasinées de nouvelles valeurs.
Une variable est alphanumérique ou numérique, suivant que la première valeur qui lui est attribuée est un chiffre ou un mot. Une variable qui est utilisée pour contenir un chiffre ne pourra être utilisée par la suite uniquement que pour contenir d'autres valeurs numériques.
De la même manière, une variable alphanumérique ne pourra être utilisée dans un contexte numérique (c'est-à-dire qu'on ne pourra pas, par exemple, additionner une valeur numérique à celle-ci). Le **TYPE** de la variable demeure inchangé pour tout l'ALGORITHME, par contre ses **valeurs** pourront changer.
Supposons maintenant que nous voulions écrire un ALGORITHME qui attribue à la variable C le maximum entre deux valeurs numériques, contenues dans les variables A 1 et A 2.




```
SI A1>A2
ALORS C=A1
AUTREMENT C=A2
FIN_SI
Imprimez C
```

L'ALGORITHME se complique si, au lieu des variables A1 et A2, nous avons 3 variables: A1, A2 et A3.

```
1   SI A1>A2
2   ALORS
3      SI A1>A3
4      ALORS C=A1
5      AUTREMENT C=A3.
       FIN_SI
6   AUTREMENT
7      SI A2>A3
8      ALORS C=A2
9      AUTREMENT C=A3.
10  FIN_SI
```

L'ALGORITHME se lit en observant les décalages à l'intérieur, dans l'écriture, qui permettent de comprendre que les actions 3-4-5 sont effectuées uniquement si A1>A2, tandis que les actions 7-8-9 sont effectuées uniquement si A2>A1 (la branche de AUTREMENT).
Que se passe-t-il si les variables deviennent 4 ou 5 ou 100? Le programme s'allongera démesurément. Notez également que les noms de variables successifs seront A4, A5... etc., parce que c'est, au fond, comme si ces noms étaient insérés dans une liste de nombres tous égaux, avec le petit chiffre à la fin qui indique qu'il s'agit du premier, du deuxième, du troisième, du dernier de la liste.
Pour résoudre cette difficulté, nous sommes autorisés à écrire en pseudocode A(1), A(2),... A(100) pour indiquer le premier, deuxième, et... centième élément d'une liste de 100 variables. De ce fait, le programme précédent devient (version I):

```
SI A(1) > A(2)
ALORS C=A(1)
AUTREMENT C=A(2)
FIN_SI
```

Jusqu'ici, nous n'avons rien gagné à utiliser les VARIABLES avec INDICE. Mais maintenant, pensez qu'il est possible d'écrire entre parenthèses non seulement une valeur explicite (p.e. A(20)) mais AUSSI UN NOM DE VARIABLE (p.e. A(N)).
Dans ce cas, nous pourrons nous référer au énième élément d'une liste, en mettant entre parenthèses le nom de la variable qui contient la valeur de l'INDICE. Les deux ALGORITHMES suivants sont équivalents:

```
a) A(3) = B(3)+C(3)
b) N=3
   A(N) = B(N)+C(N).
```

La série d'instructions de l'exemple b) aboutit au même résultat que celui de l'instruction de l'exemple a).
Nous sommes maintenant en mesure d'écrire l'ALGORITHME qui trouve le maximum entre les 20 nombres contenus dans les variables A(1), A(2)...,A(20) avec le simple ALGORITHME:

```
I=1
C=A(1)
REPETEZ JUSQU'A CE
QUE I>2∅
```

```
    SI C<A(I)
    ALORS
       C=A(I)
    AUTREMENT
    FIN_SI
       I=I+1
  FIN_REPETEZ
  Ecrivez C
```

A la fin de l'ALGORITHME, C contiendra le maximum entre les 20 valeurs de variables numériques A(1)... A(20).
Comment l'ALGORITHME fonctionne-t-il? Tout d'abord, on attribue à la variable I la valeur 1, et on attribue à C la valeur du premier élément de la liste, (c.à.d.A(1)). Cette première attribution a uniquement pour but de commencer le programme en donnant à C la valeur maximale rencontrée ici; effectivement, si A(1) était la seule variable existante, la valeur maximale serait justement la valeur de A(1). Si on attribuait à C la valeur initiale ∅, l'ALGORITHME ne restituerait pas le maximum dans le cas où les valeurs de la liste seraient toutes négatives (des erreurs de ce type sont fréquentes: on dit en jargon que l'ALGORITHME est "instable au contour". Cette phrase signifie qu'il existe des limites dans lesquelles l'ALGORITHME a un comportement anormal: dans notre cas, si les nombres sont tous positifs ou nuls, l'ALGORITHME restitue la valeur maximale; si, par contre, ils étaient négatifs, il restituerait ∅).
Mais revenons à notre ALGORITHME. La valeur de C est confrontée, à l'intérieur du cycle, à chacune des 20 valeurs des variables A(1)... A(20).
A chaque nouvelle confrontation, si la variable A(I) est plus grande que C, on fait une nouvelle attribution, par laquelle on ré-impose C avec la valeur de A(I): de cette manière, à chaque passage, la variable C est mise à jour à la valeur nominale rencontrée.
On peut observer que l'ALGORITHME reste identique indépendamment du nombre des variables dans la liste: il suffit de modifier la limite numérique dans le prédicat de contrôle (c.à.d. changer le 20 avec le nombre approprié) et l'ALGORITHME fonctionne encore.
En pseudocode, de même que la simple citation d'un nom de variable valait comme déclaration d'existence de cette variable, la simple citation en une phrase de pseudocode d'une variable avec indice implique la déclaration d'une telle variable avec indice. Nous serions donc contraints de toujours utiliser, pour la suite du programme, le nom de cette variable en mettant entre parenthèses soit une valeur explicite, soit le nom d'une variable numérique.
Dans l'exemple du maximum dans une liste da variables, la variable I, qui sert d'indice à la liste, est utilisée pour "balayer" la liste: elle est mise à 1 avant la répétition et puis augmentée de 1 au cours du cycle. Cette utilisation de la variable est tellement fréquente qu'on a ajouté,



comme figure additionnelle, la construction syntaxique suivante, appelée répétition énumérative:

```
REPETEZ POUR I
QUI VARIE DE 1 à 1Ø
 Action 1
 Action 2
    :
    :
FIN_REPETEZ
I = 1
REPETEZ JUSQU'A CE
QUE I > 1Ø
 Act. 1
 Act. 2
 I = 1 + 1
FIN_REPETEZ
```

La répétition énumérative est exactement l'équivalent de la répétition avec une variable de balayage.
Le programme "maximum entre 20 nombres" peut donc être écrit:

```
C = A(1)
REPETEZ POUR I
QUI VARIE DE 1 à 2Ø
   SI A(I) > C
   ALORS
      C = A(I)
   AUTREMENT
   FIN_SI
FIN_REPETEZ
```

Vous remarquerez que l'on économise 2 instructions: celle qui attribue 1 à la variable I, et celle qui augmente de 1 la variable I: les deux instructions sont sous-entendues par la phrase: REPETEZ POUR I QUI VARIE de 1 à 20, qui implique déjà que la valeur initiale de 1 sera 1 et que la variable sera augmentée de 1 à chaque cycle.
Il convient d'utiliser toujours la répétition énumérative avec augmentation de 1 même lorsque l'on veut produire diverses successions de nombres: il est plus clair d'utiliser la répétition avec progression de 1 à chaque fois. Par exemple, faisons écrire les 50 premiers nombres impairs?
Voici l'ALGORITHME:

```
N = 1
REPETEZ POUR J
QUI VARIE DE 1 à 5Ø
 N = N + 2
 ECRIVEZ N
FIN_REPETEZ
```

Avec cet ALGORITHME nous obtenons 50 passages de l'instruction N = N + 2 et nous obtenons donc 50 nombres impairs.
La variable J, déclarée implicitement au moment où s'écrit le REPETEZ POUR, peut être utilisée à l'intérieur du cycle en n'importe quelle combinaison.
Par exemple, l'ALGORITHME précédent est également équivalent au suivant:

```
REPETEZ POUR J
QUI VARIE DE 1 à 5Ø
 N = (2 x J)—1
 Ecrivez N
FIN_REPETEZ
```

Dans cet ALGORITHME, les nombres impairs sont obtenus en multipliant les valeurs successives de J par 2 et en soustrayant 1. Cette fois aussi, nous obtenons les 50 premiers nombres impairs.

Comme nous le voyons, avec la répétition énumérative nous pouvons obtenir beaucoup d'effets intéressants, particulièrement en jouant avec la variable du cycle, c'est-à-dire la variable que nous utilisons pour balayer la liste. Dans l'exemple 4, nous voyons une utilisation de la variable IX servant à produire l'effet de voir écrit précisément A(16), c'est-à-dire le nom de la variable qui contient le numéro maximum.

Le nom "variable avec indice" n'est pas le seul nom utilisé pour décrire cette catégorie de variables: dans le milieu scientifique, on dit également "vecteur". Etant donné que le BASIC est né dans le milieu scientifique, les variables avec indice en basic s'appellent vecteurs et le nombre d'éléments maximum du vecteur s'appelle "dimension du vecteur".
En BASIC, un vecteur est "déclaré" en tête du programme, en en déterminant la dimension; par exemple:

```
1 DIM B (51∅)
```

définit un vecteur numérique de nom B et de longueur 51∅. En d'autres termes, on définit une liste de 51∅ éléments, chacun étant indiqué par le nom B (J), où J peut avoir comme valeur de 1 à 51∅.
Vous observerez aussi que, lorsqu'une variable est utilisée entre parenthèses pour indiquer un élément d'un vecteur, cette variable a la "fonction d'un indice".
De ce fait, un "indice" n'est autre qu'une variable numérique quelconque, qui est utilisée en fonction d'indice, mais elle n'a aucune autre caractéristique.
Nous terminons ici l'article d'aujourd'hui. Essayer d'introduire les programmes d'exemple dans votre ordinateur, et changez les valeurs numériques des attributions de test, en y mettant les valeurs de votre choix.
Observez le comportement du programme de l'exemple 2 par rapport à celui de l'exemple 1, lorsqu'on est en présence de valeurs négatives.
Puis introduisez le programme de l'exemple 4 et essayez de le faire tourner. Vous aurez la satisfaction de voir imprimé le nom de la variable qui contient l'élément maximum du vecteur. Comment cela fonctionne-t-il? Essayez de le découvrir par vous-même.
Dans le prochain numéro nous parlerons de variables avec doubles indices (matrices) et des instructions de I/0 simples (Input/Output = Entrées/Sorties). Nous etudierons un programme d'exemples pour l'utilisation de ces variables en un simple ALGORITHME pour la production d'un graphique à barres.

M.S.




```
10 DIM A(20)
20 A(1)=14:A(2)=34:A(3)=30:A(4)=20
30 A(5)=33:A(6)=24:A(7)=54:A(8)=33
40 A(9)=20:A(10)=28:A(11)=24:A(12)=34
60 A(17)=94:A(18)=53:A(19)=10:A(20)=58
70 I=1
75 C=A(I)
80 OF I > 20 GOTO 120
90 IF C < A(I) THENC=A(I)
100 I=I+1
110 GOTO80
120 REM FIN-REPETEZ
130 PRINT"LA VALEUR MAXIMALE EST";C

READY.
```

Exemple 1. Programme qui extrait le maximum entre 20 nombres.

```
10 DIM A(20)
20 A(1)=14:A(2)=34:A(3)=30:A(4)=20
30 A(5)=33:A(6)=24:A(7)=54:A(8)=33
40 A(9)=20:A(10)=28:A(11)=24:A(12)=34
50 A(13)=73:A(14)=60:A(15)=48:A(16)=14
60 A(17)=94:A(18)=53:A(19)=10:A(20)=58
70 I=1
75 C=0
80 IF I>20 GOTO120
90 IF C<A(I)THENC=A(I)
100 I=I+1
110 GOTO80
120 REM FIN-REPETEZ
130 PRINT"LA VALEUR MAXIMALE EST";C

READY.
```

Exemple 2. Le même programme qu'auparavant dans lequel C est initialisé à ∅. Il ne permet pas de trouver le maximum si les nombres sont négatifs.

```
10 FOR J=1TO50
20 N=(J*2)-1
30 PRINTN;
40 NEXTJ

READY.

  1   3   5   7   9   11   13   15   17   19   21   23   25
  27   29   31   33   35   37   39   41   43   45   47   49
  51   53   55   57   59   61   63   65   67   69   71   73
  75   77   79   81   83   85   87   89   91   93   95   97   99
 READY.
```

Exemple 3. Programme qui produit les 50 premiers nombres impairs, avec exemple de output.

```
10 DIM A(20)
20 A(1)=14:A(2)=15:A(3)=20:A(4)=28
30 A(5)=34:A(6)=24:A(7)=45:A(8)=23
40 A(9)=32:A(10)=20:A(11)=55:A(12)=25
50 A(13)=22:A(14)=24:A(15)=30:A(16)=73
60 A(17)=22:A(18)=33:A(19)=20:A(20)=40
70 C=A(I)
75 FOR I=1TO20
80 IFC<A(I)THENC=A(I):IX=I
90 NEXTI
100 PRINT"LE MAXIMUM VAUT ";C;" QUI CORRESPOND A
                              LA VARIABLE A(";IX;")"

READY.
```

Exemple 4. Programme qui extrait la valeur maximale dans la liste A(N); et écrit aussi le nom de la variable qui le contient.

LES TOUCHES FONCTIONNELLES DU VIC-20

Quand nous regardons notre ordinateur, nous remarquons quatre touches de couleurs différentes placées sur le côté droit de la machine. Ces curieux "boutons" s'appellent "touches fonctionnelles". Leur rôle est de fournir une série d'entrées à partir du clavier, bien définies et différentes entre elles.
Il n'est pas exclu qu'à première vue elles puissent sembler faciles à comprendre et à utiliser. Cependant, si elles ne sont pas insérées dans une logique de programme correcte, elles peuvent engendrer quelques confusions.
Nous jugeons donc nécessaire de faire quelques commentaires à ce sujet car la question que l'on peut se poser le plus souvent est: comment utiliser correctement ces touches?

La commande INPUT

Comme cela est spécifié dans les revues BASIC, si le programme contient une instruction de ce type, l'exécution est interrompue jusqu'à ce qu'une information apparaisse en vidéo et que l'on appuie ensuite sur la touche Return.
Si l'on appuie alors sur l'une des touches fonctionnelles de F1 à F8, la première impression est que ces touches ne semblent pas être commandées par notre ordinateur.
Le problème est que l'instruction INPUT est structurée de manière à prélever les informations du vidéo, mais si l'on appuie sur l'une des "Function keys", rien n'apparaît sur l'écran.
L'explication de tout ceci se trouve dans la logique d'architecture du calculateur dans la mesure où, lorsqu'on appuie sur l'une des touches fonctionnelles, l'ordre est reçu par la machine et mémorisé dans le "Buffer" du clavier (le buffer du clavier n'est autre que des emplacements de mémoire précis qui mémorisent les informations introduites).
Au moment où le programme reconnaît une instruction INPUT, l'exécution est bloquée et un point d'interrogation suivi du curseur clignotant apparaît sur le vidéo.
Tant qu'une information n'aura pas été introduite et que l'on n'aura pas appuyé sur la touche Return, le programme ne recommencera pas à tourner.
Au cours de ce processus, le buffer du clavier enregistre les informations introduites mais se vide immédiatement et les caractères passent d'abord sur le vidéo et, après avoir appuyé

sur la touche Return, sont reçus par le programme. Etant donné que, dans ce cas particulier, F1, F2..F8 n'ont pas de caractère équivalent pour apparaître en vidéo, l'information est perdue. De ce fait, en appuyant sur Return, le programme n'est pas en mesure de recevoir quelque chose puisqu'aucun caractère n'apparaît sur le vidéo.
En d'autres termes, la commande Input n'est pas en mesure de reconnaître que l'on a appuyé sur une touche fonctionnelle.
Il existe cependant une manière de contourner l'obstacle, mais elle n'est pas très élégante: il suffit de mettre la variable objet de l'Input entre guillemets (les") de manière à ce que les touches fonctionnelles soient reconnues elles aussi grâce à l'apparition d'un symbole graphique sur le vidéo.
Par exemple:

```
1∅ INPUT "B$"
```

La commande GET

Examinons maintenant cette instruction que l'on estime être un meilleur instrument pour gérer les FUNCTION KEYS.
La principale caractéristique de cette commande est que les informations sont prélevées directement par le buffer du clavier, permettant ainsi au programme de reconnaître sans problème qu'une touche fonctionnelle a été enfoncée. Essayons maintenant de nous poser cette question: "Comment faire pour qu'un programme, au cours de l'exécution, puisse gérer les touches fonctionnelles?"
La réponse n'est pas très difficile et, même si nous rencontrons quelques petits problèmes de nature graphique, nous essaierons d'arriver à la solution la meilleure et la plus facile à travers la construction d'un programme bref.
Commençons tout d'abord par introduire dans notre ordinateur les instructions suivantes:

```
1∅ GET X$
2∅ IF X$ = "
```

Arrivés à ce stade, nous nous trouvons au milieu de la ligne 20 et nous venons juste de tracer les guillemets "lisez". Si nous appuyons maintenant sur la touche fonctionnelle F1 nous voyons apparaître un curieux symbole graphique. Ce n'est autre que l'équivalent sur le vidéo de la touche fonctionnelle F1 et il apparaît sous la forme d'une barre horizontale en négatif.
Nous terminons l'instruction 20 de cette manière et sur une seule ligne:

```
2∅ IF X$ = "f1" THEN PRINT
     "VOUS AVEZ APPUYE
     sur F1"
```

Faisons attention à taper la touche F1 entre guillemets de manière à ce que le symbole graphique correspondant apparaisse sur le vidéo et non les touches F et 1 comme s'il s'agissait de deux caractères.
Cette possibilité permettra à

notre ordinateur de reconnaître que l'on a appuyé sur la touche F1 et d'imprimer sur le vidéo le message objet de l'instruction PRINT.
De la même manière, l'instruction suivante servira à gérer F2 (sur une seule ligne)

```
3∅ IF X$ = "f2" THEN PRINT
   "VOUS AVEZ APPUYE
   sur F2"
```

Pour obtenir le symbole graphique de F2 il faut appuyer en même temps sur la touche SHIFT et sur F1.
En répétant ces instructions pour toutes les autres touches de F3 à F8, on obtiendra:

```
4∅ IF X$ = "f3" THEN PRINT
"VOUS AVEZ APPUYE sur F3"
5∅ IF X$ = "f4" THEN PRINT
"VOUS AVEZ APPUYE sur F4"
6∅ IF X$ = "f5" THEN PRINT
"VOUS AVEZ APPUYE sur F5"
7∅ IF X$ = "f6" THEN PRINT
"VOUS AVEZ APPUYE sur F6"
8∅ IF X$ = "f7" THEN PRINT
"VOUS AVEZ APPUYE sur F7"
9∅ IF X$ = "f8" THEN PRINT
"VOUS AVEZ APPUYE sur F8"
```

Il ne reste plus qu'à ajouter la dernière instruction:

```
1∅∅ GOTO 1∅
```

de manière à ce que le programme exécute une boucle, c'est-à-dire qu'il continue à tourner sans s'arrêter.
En appuyant sur RETURN, notre calculateur effectuera cette série d'instructions et nous verrons qu'en appuyant sur l'une des touches fonctionnelles, la description correspondante apparaîtra sur le vidéo.
Nous avons donc créé un programme capable de reconnaître sans problème l'introduction d'une FUNCTION KEY.

Une manière plus satisfaisante de gérer la situation

Ces étranges caractères graphiques qui apparaissent sur le vidéo au cours de l'écriture du programme peuvent souvent créer une confusion. Vous remarquerez qu'ils sont difficiles à lire et que l'on pourrait les confondre entre eux. Dans ce cas, à la suite de l'impression du programme sur le papier, il peut arriver que l'imprimante ne reconnaisse pas le caractère ou imprime un symbole certainement peu compréhensible. Il peut être utile également de gérer le tout à travers un petit stratagème qui nous permettra de remédier à ce problème de nature graphique.
Il existe en BASIC une fonction déjà établie qui convertit n'importe quel symbole graphique en sa valeur correspondante ASCII (American standard Code for Information Interchange).
Il s'agit de la fonction ASC.
Reprenons notre programme et essayons de le modifier de la manière suivante (sur une ligne):

1∅ GET X$: IF X$ = " "
= GOTO 1∅

Introduisons aussi:

15 X = ASC (X$)

nous avons seulement introduit en premier un contrôle dans l'instruction GET de façon que le programme exécute les instructions successives seulement si on appuie sur une touche quelconque et, à travers la fontion ASC dans l'instruction 15, la valeur ASCII du caractère sur lequel on a appuyé se trouve attribuée à la variable numérique X.

Il est logique qu'il existe aussi une valeur correspondante ASCII pour les touches fonctionnelles.

Nous pouvons maintenant faire la liste de toutes ces valeurs mais, afin de mieux comprendre ce type de fonction, nous préférons vous dire de quelle manière vous pouvez trouver, de vous-mêmes, ces valeurs numériques.

Supposons que nous voulions trouver la valeur ASCII de F1.

Sans craindre de perdre de la mémoire le programme que nous avons écrit, nous pouvons introduire:

PRINT ASC ("f1").

en faisant toujours attention de taper sur la touche fonctionnelle F1 et non F et 1 comme s'il s'agissait de deux caractères.

Notre ordinateur fera apparaître immédiatement sur le vidéo le numéro 133 qui correspond précisément à la valeur ASCII de F1.

Nous pouvons répéter de la même manière cette instruction pour les huit touches et nous verrons apparaître les valeurs correspondantes.

Mais revenons à notre programme, dans lequel nous avons deux nombres et non plus des symboles graphiques. Les instructions successives seront:

2∅ IF X = 133 THEN PRINT
"VOUS AVEZ APPUYE
SUR F1"
etc.

Cette solution est certainement la plus sûre et la plus élégante pour gérer les touches fonctionnelles qui, nous l'espérons, vous semblent maintenant plus compréhensibles qu'auparavant.

Pour terminer, je vous propose un simple programme, créé pour simuler des devinettes posées par votre ordinateur.

A vous la satisfaction de l'adapter aux demandes les plus spécifiques.

C.G.

EXEMPLE

1∅	texte de la demande
2∅	texte des quatre réponses possibles
3∅	position de la réponse exacte
4∅	idem comme 1∅
5∅	idem comme 2∅
6∅	idem comme 3∅
7∅	idem comme 1∅
8∅	idem comme 2∅

9∅	idem comme 3∅
1∅∅	dernière donnée à lire pour la fin du programme
11∅	lit le texte de la demande, si la donnée FIN est lue, le programme se termine.
12∅	imprime le texte de la demande
13∅-16∅	lit par l'instruction DATA le texte des quatre réponses possibles et imprime le contenu sur le vidéo.
18∅	lit par l'instruction DATA le chiffre qui indique la position de la réponse exacte.
19∅	reconnaît l'introduction par le clavier
2∅∅	convertit le caractère graphique en valeur ASCII
21∅	contrôle que la valeur ASCII correspond bien aux valeurs relatives des touches fonctionnelles
22∅	imprime sur le vidéo celle des touches sur laquelle on a appuyé. (NOTE: la variable X1 sert à gérer la séquence alternée des valeurs ASCII des touches fonctionnelles)
23∅	vérifie que la réponse est exacte en confrontant la valeur introduite avec celle qui est lue par le DATA
25∅	fin LOOP

```
READY

1∅ DATA Où se trouve la France?
2∅ DATA Asie, Europe, Amérique du Sud, Afrique
3∅ DATA 2
4∅ DATA De qui est la Divine Comédie?
5∅ DATA Boccace, Dante, Stendhal, Pascal
6∅ DATA 3
7∅ DATA Qui a découvert l'Amérique?
8∅ DATA Colomb, Galilée, Einstein, Reagan
9∅ DATA 1
1∅∅ DATA FIN
11∅ READ B$: IF B$ = "FIN" then 26∅
12∅ PRINT B$
13∅ READ A$: PRINT "F1-"; A$
14∅ READ A$: PRINT "F3-"; A$
15∅ READ A$: PRINT "F5-"; A$
16∅ READ A$: PRINT "F7-3; A$
17∅ PRINT "quelle est votre réponse?"
18∅ READ C
19∅ GET X$: IFX$ = ""THEN19∅
```

```
2∅∅ X = ASC (X$)
21∅ IFX<133 ORX>136GOTO19∅
22∅ X = X-132:X1=(X*2)-1:PRINT"Touche F"; X1
23∅ IFX = CTHENPRINT "Réponse exacte!": GOTO 25∅
24∅ PRINT "DESOLE', VOUS VOUS ETES TROMPE"
25∅ GOTO 11∅
26∅ STOP

READY
```

LE BASIC C'EST QUOI AU JUSTE?

A n'en pas douter, le BASIC du Spectrum, avec ses quelque 90 fonctions, est un des langages les plus complets dont on dispose pour l'ordinateur domestique.
Mais en réalité, qu'est-ce que le BASIC? Question qui équivaut à une autre question: qu'est-ce qu'un langage de programmation?
Avant de répondre, il faut tout d'abord préciser que l'ordinateur ou, mieux encore, l'UC (unité centrale) comprend seulement des nombres binaires composés de 8 bits. A ces nombres correspondent des signaux électriques déterminés qui sont transmis aux commandes du microprocesseur. Chaque nombre - toujours compris, on le rappelle, entre 0 et 255, équivaut à une opération élémentaire accomplie par l'UC de l'ordinateur.
Dans la MEM (ROM) du Spectrum (et de la plupart des ordinateurs) se trouve un long programme entièrement écrit en langage machine. Ce programme qui englobe chacune des activités de l'ordinateur est appelé SYSTEME OPERATIONNEL et il comprend dans bien des cas un interpréteur.
L'interpréteur n'est jamais qu'une partie du programme en langage machine qui se charge de déchiffrer les instructions en BASIC autrement incompréhensibles pour l'UC et qui donne pour chacune de celles-ci une série d'instructions en code machine. Vous réaliserez donc que plus les instructions à disposition sont nombreuses, plus l'interpréteur mettra de temps à les reconnaître. Ceci explique la lenteur relative du BASIC du Spectrum qui possède, comme on le disait, une gamme étendue de fonctions.
De plus, chaque fois que vous faites tourner un programme, les instructions qu'il contient doivent être redéchiffrées. Pour pallier cet inconvénient, des interpréteurs compilateurs ont été mis au point, qui après avoir exécuté une seule fois un programme le codifient directement en langage machine. Bien entendu, après cette première opération, l'interprétation n'est plus nécessaire dans la mesure où l'ordinateur reconnaît le programme tel quel.
Malheureusement, cette solution a aussi ses inconvénients: après la compilation, il est impossible de corriger ou modifier le programme et par conséquent, en cas d'erreurs, il faut souvent le réécrire entièrement.
C'est la raison pour laquelle la plupart des ordinateurs incorporent un interpréteur qui permet au programmeur de corriger facilement et en toute sécurité ses erreurs et de modifier en cours de route ses programmes.

Massimo Cellini

AUGMENTONS LA VITESSE D'ÉCRITURE DE VOS PROGRAMMES BASIC

Peut-être ce programme sera-t-il le dernier que vous écrirez de manière traditionelle.
Cet article vous présente la manière dont vous pourrez introduire dans votre ordinateur les instructions BASIC en appuyant seulement sur une lettre et sur la touche SHIFT.
Il est possible d'introduire, aussi bien dans le Commodore 64 que dans le VIC 20, certaines instructions BASIC abrégées en appuyant sur la touche SHIFT puis sur une touche correspondant à un symbole graphique, mais combien de signes bizarres et incompréhensibles apparaissent sur le vidéo, et quelle fatigue pour se les rappeler tous!
Ne craignez rien, grâce à tous ces programmes construits pour le Commodore 64 et le VIC 20, nous pourrons écrire les instructions BASIC en appuyant seulement sur la touche SHIFT en même temps que sur la première lettre de l'instruction et, comme par magie, l'instruction correspondante apparaîtra en entier sur le vidéo.
Par exemple, en appuyant sur la touche SHIFT et en même temps sur la touche A, apparaît l'instruction ASC; SHIFT-B fera apparaître STEP, SHIFT-C CHR$ et ainsi de suite pour toutes les indications dont voici la liste ci-dessous:

* * *

Pour conserver et utiliser le programme, il faut l'écrire dans l'ordinateur, le conserver et donner le RUN après la sauvegarde. Si une erreur de frappe a été commise dans les instructions DATA, un message d'avertissement apparaît. Au cas où tout fonctionne correctement, il faudra, pour mettre le programme en route, taper l'instruction SYS 52557 pour le Commodore 64 et SYS 7501 pour le VIC-20.
Dès lors, tout est prêt pour écrire un programme BASIC à la vitesse de la lumière! Toute plaisanterie mise à part, essayez de donner une instruction BASIC avec la touche SHIFT et vous verrez que, sous le message READY, le mot tout entier apparaît.

Naturellement, il est toujours possible de taper l'instruction entièrement, vous devez même le faire si celle-ci n'est pas dans la liste. Veillez à taper ensuite l'ordre NEW avant d'écrire votre programme BASIC.
Il faut ensuite vous rappeler que tout ordre graphique nécessitant la touche SHIFT est toujours autorisé, pourvu qu'il soit tapé entre guillemets (lire").

NOTE: Si on tape une deuxième fois SYS 52557, le programme d'abréviation des instructions se trouve désactivé sans être cependant effacé de la mémoire. On obtient le même résultat en tapant RUN/STOP et RESTORE. C'est seulement en éteignant l'ordinateur que la mémoire sera déchargée.

LE JOYSTICK DU VIC-20

Aussi bien dans le Vic-20 que dans le Commodore, le joystick est géré par les valeurs contenues dans deux BIT de mémoire. Dans ce but, l'instruction PEEK permet d'examiner une location spécifique et de voir quelle est la valeur présente dans celle-ci.
Le Vic 20 est construit de manière à gérer un seul joystick à travers deux BIT de contrôle.
La location 37137 sert au contrôle des mouvements vers le Haut, le Bas, la Gauche et la touche FIRE. La location 37152 contrôle le mouvement vers la droite.
Le Commodore 64, par contre, est construit de manière à gérer deux joysticks et, à l'opposé du VIC-20, chaque joystick est contrôlé par une seule location de mémoire. La porte A est gérée par la location 56320 et la porte E par la location 56321.
Voyons maintenant de quelle manière il est possible de vérifier le mouvement du joystick. Après avoir branché l'appareil, nous pouvons écrire un bref programme afin de vérifier ce que nous venons de dire.
Le programme ne fait que prélever le contenu de la location de mémoire correspondante et imprimer le contenu sur le vidéo.
Pour le VIC-20, les instructions sont:

Pour le Commodore 64:

Après avoir donné le RUN du programme, si nous essayons de bouger l'un des joysticks, il est possible de visualiser sur l'écran de quelle manière les valeurs changent selon le mouvement imprimé au joystick.
De cette manière, il est possible de construire des programmes qui utilisent les joysticks à travers la gestion des valeurs contenues dans les locations de mémoire correspondantes.
C'est donc la fonction PEEK qui, fondamentalement, peut être intégrée avant une instruction IF...THEN qui permet au programme d'activer des instructions dépendant de la valeur prélevée dans la location du contrôle du joystick.
Se pose une dernière question, et non la moindre, concernant les BIT de contrôle. Il est nécessaire de se rappeler que ces locations de mémoire peuvent également contenir des valeurs différentes de celles concernant le mouvement du joystick. Il est donc beaucoup plus sûr et efficace de prélever avec l'instruction PEEK uniquement les informations concernant les BIT de contrôle du joystick. La technique consiste à insérer, après l'instruction PEEK, un AND suivi de la valeur utilisée par ce type de gestion.
Par exemple, posons le problème du contrôle si on a appuyé sur la touche FIRE du joystick. La touche FIRE met le BIT 4 du porte-jeux correspondant (1 ou 2) à zéro; il faut donc prélever

uniquement ce bit et en tester l'état.
Pour ce faire, si on examine la porte 1, on exécute l'instruction suivante:

X = PEEK (5632 ∅) AND 16

X contiendra la valeur du BIT 4, c'est-à-dire 16 si le bit est à 1, ou zéro si le bit est à zéro, donc si x est égal à zéro, le joueur aura tiré.

Naturellement ceci n'est qu'une introduction à l'usage du joystick, d'autres méthodes et d'autres exemples sophistiqués d'utilisation seront étudiés dans les prochains numéros.

COMMODORE & SINCLAIR: LES NOUVEAUTES

Le boom de l'informatique domestique a provoqué de gros conflits d'intérêt sur le plan international.
La bataille ne se joue pas seulement autour des machines, qui ne sont jamais qu'une tranche de ce faramineux gâteau. Une bonne partie des recettes provient des ventes de logiciels, qui ne cessent d'augmenter malgré les progrès de la piraterie.
Outre les constructeurs de hardware et les entreprises de logiciels, on trouve aussi dans la mêlée un nombre impressionnant de maisons indépendantes spécialisées (ou qui se disent telles) dans la production de périphériques en tous genres pour ordinateurs familiaux et individuels et qui sont prêtes à exaucer le moindre de vos désirs, que vous leur demandiez une simple extension de mémoire, un plotter sophistiqué ou encore un super joystick mental.
Commodore et Sinclair étant les deux marques leaders sur le marché européen, il n'est pas étonnant que l'on trouve une quantité de périphériques construits pour les ordinateurs produits par ces deux maisons.
Dans cet article, on entreprend une analyse générale des dernières nouveautés disponibles à l'usage des ordinateurs Spectrum et Vic 20. Sans trop s'attarder sur les différents modèles existants, on se limitera à présenter rapidement les caractéristiques et les applications possibles de ces produits.
Commençons par parler du périphérique qui est probablement le plus répandu chez les utilisateurs d'ordinateur individuel: l'imprimante.
En général, quand on achète un ordinateur on n'éprouve pas immédiatement le besoin de voir reproduits sur le papier les travaux réalisés par celui-ci mais on a tendance à considérer l'imprimante comme un éléments superflu. Bien vite toutefois, on se rend compte que celle-ci est un accessoire très utile voire indispensable.
Contrairement au Vic, le Spectrum dispose d'un unique raccord pour extension (qui plus est, non standard), qui représente le seul moyen de connecter l'ordinateur aux divers périphériques. Il est par conséquent impossible d'adapter au Spectrum des imprimantes réalisées suivant les standards RS-232 ou Centronics, à moins de recourir à de coûteuses interfaces qui, à leur tour, nécessitent

souvent des logiciels supplémentaires. Chaque accessoire doit donc être conçu expressément pour le raccord avec cet ordinateur.
Bien que la Sinclair ait produit de nombreuses extensions qui élargissent considérablement les possibilités du Spectrum, ceci ne suffit pas à satisfaire les exigences des utilisateurs les plus "évolués" (en matière de programmation), qui ne sauraient se contenter d'une lente et bruyante imprimante thermique, celle-ci s'avérant par contre idéale pour l'utilisateur moyen qui dispose d'un petit budget (cas des plus répandus chez les jeunes amateurs).
C'est donc avec un soupir de soulagement que l'on salue l'arrivée de la GP-50s, la "petite dernière" de SEIKOSHA en version "spectrumisée". La GP-50s est une véritable imprimante à impact qui utilise un papier blanc normal de 5". Sa vitesse d'impression est de 35 CPS pour des lignes de 32 caractères.
La version GP-50A utilise par contre les standards Centronics et imprime des lignes de 45 caractères. Elle peut donc facilement être reliée à la plupart des ordinateurs.
A propos des supports papier, une nouvelle imprimante graphique à haute résolution et 3 ou 7 couleurs, entièrement compatible avec le VIC 20, a été lancée tout dernièrement. Il s'agit de la Okimate 10 qui, paraît-il, est vendue aux Etats Unis à des prix super intéressants. Espérons que quelqu'un se chargera au plus vite de l'importer en France.
La Commodore aussi a récemment présenté une nouvelle imprimante mais pour le moment l'information est rare. Attendons d'en savoir plus.
A part les imprimantes, les périphériques les plus demandés sont les mémoires de masse d'accès rapide et sûr (disquettes ou floppy disk).
Pour le Spectrum une interface a été mise au point qui permet d'assurer une communications avec des lecteurs de disquette normale de 5 1/2". Bien que le prix soit nettement supérieur à celui des fameux microdrives de la Sinclair, voilà une nouveauté qui sera sûrement bien accueillie par ceux qui souhaitaient faire un usage professionnel de cet ordinateur mais pour qui les cartouches tristement célébres constituaient un obstacle insurmontable, vu notamment leur prix prohibitif.
Les lecteurs de disquettes du VIC présentent pour leur part, des problèmes tout à fait différents. Bien qu'ils utilisent des disquettes normales, leur vitesse de chargement est hélàs bien trop faible ce qui limite grandement les possibilités d'application pour un programme comportant de fréquentes opérations de chargement et déchargement sur disque.
La Commodore s'est attaquée au problème et a imaginé pour le résoudre deux nouveaux modèles de lecteurs: le 481 et le 1001 qui sont nettement plus rapides que leur prédécesseurs.
Voilà. Notre revue des nouveautés est terminée pour le moment. Ne manquez pas notre prochain rendez-vous car le numéro à venir vous proposera une foule d'informations intéressantes sur les ordinateurs domestiques.

COMMENT LE SPECTRUM MEMORISE LES DONNEES

Vous ne vous serez peut-être jamais demandé comment le Spectrum (ou n'importe quel autre ordinateur à 8 bits) emmagasine et élabore les informations qui lui sont données au moyen du clavier.
En réalité, ce qui se passe à l'intérieur du Spectrum n'est pas bien compliqué. Cela est même d'une simplicité étonnante.
Comme on l'a vu, le Spectrum est un ordinateur à 8 bits, ce qui veut dire que n'importe quelle donnée doit être stockée sous forme d'OCTET c'est-à-dire de nombres binaires composés précisément de 8 bits.
Bien qu'à première vue le système binaire puisse apparaître très éloigné du système décimal utilisé couramment, il suffit d'approfondir la question pour découvrir qu'en réalité tous les systèmes de numération se ressemblent.
En utilisant le système décimal, pour calculer la valeur maximale représentable avec un nombre déterminé de chiffres, il suffit d'élever 10 (la base du système) à la puissance du nombre de chiffres employés et de soustraire 1 du résultat.
Par exemple, pour trouver le plus grand nombre entier représentable par 3 chiffres, l'on procède de la façon suivante:

10 ↑ 3 = 1000 — 1 = 999

Simple, n'est-ce pas?
La même marche vaut pour le système binaire: il suffit de changer la base qui ne sera plus de 10 mais de 2. Sachant donc qu'un OCTET (byte) est composé de 8 chiffres (bits), on peut obtenir la plus grande valeur entière que celui-ci peut contenir:

2 ↑ 8 = 256 — 1 = 255

Chaque bit peut donc contenir une valeur comprise entre 0 et 255.
Comment alors l'ordinateur peut-il emmagasiner des valeurs allant jusqu'à 65535 dans le cas d'un système de 64 K (comme le Spectrum: 16 ROM + 48 RAM = 64)?
Facile! Il peut le faire en réunissant 2 OCTET de façon à disposer non plus de 8 mais de 16 bits pour représenter la valeur numérique.
Refaisons donc le calcul de tout à l'heure:

2 ↑ 16 = 65536 — 1 = 65535

Elémentaire non?
Un petit problème se pose tout de même: la mémoire du Spectrum étant organisée sous forme de d'OCTET, pour pouvoir stocker une valeur de 16 bits, il faut la décomposer en 2 nombres de 8 bits. Par conséquent il y aura stockage d'abord de la partie la moins significative puis de la plus significative ensuite.
C'est clair? J'espère que oui parce qu'il vous faudra assimiler parfaitement tous ces concepts pour comprendre la suite qui s'annonce palpitante dans le prochain numéro.

Massimo Cellini

ISTRUCTIONS POUR LA CASSETTE DE SOFTHEQUE n° 1 (côté VIC 20)

Pour le côté ZX Spectrum les instructions apparaîtront directement sur le video

COMMENT COMMENCER

Prendre la cassette, la mettre dans le magnétophone avec la face choisie, A ou B, vers le haut.

Allumer l'ordinateur, écrire l'instruction 'LOAD' et appuyer sur la touche 'return': l'inscription "PRESS PLAY ON TAPE" apparaîtra; elle signifie "appuyer sur la touche PLAY sur le magnétophone". Exécuter l'ordre et attendre que l'inscription "FOUND INTRODUCTION A" apparaisse sur l'écran; à ce point, appuyer sur la touche d'espacement et attendre quelques secondes, jusqu'à ce que l'écran réapparaisse avec le curseur clignotant.

Si une inscription d'erreur apparaît, ré-enrouler la cassette et recommencer les opérations.

Si tout est O.K., appuyer sur la touche "STOP" du magnétophone, écrire sur l'écran "RUN" et taper "RETURN" sur l'ordinateur; après quelques instants apparaît la première page de notre revue avec la liste des programmes contenus dans la cassette. Lorsque apparaît l'inscription "ENFONCER LA TOUCHE", appuyer sur une touche du clavier au hasard. Dès que l'écran devient bleu, avec le bord bleu-clair et les inscriptions dans la partie supérieure que vous voyez normalement en allumant l'ordinateur , écrivez de nouveau "LOAD" et appuyez sur "RETURN"; appuyez sur "PLAY" sur le magnétophone et attendez que l'ordinateur trouve le programme suivant, appuyez sur la touche d'espacement et attendez un instant jusqu'à ce que le curseur clignotant réapparaisse. Si aucune inscription d'erreur n'apparaît, appuyez sur "STOP" sur le magnétophone, formez "RUN" sur l'écran et appuyez sur "RETURN" sur l'ordinateur, afin de faire démarrer le programme. Suivez ces simples instructions chaque fois que vous voulez charger un programme du softhèque ordinateur.

Si vous aviez du mal à charger les programmes et si des inscriptions d'erreur (ex. PRINT LOAD ERROR) apparaissaient sur l'écran, vous pourriez essayer de modifier la position des deux têtes à l'aide d'un tournevis introduit dans le trou situé dans la partie supérieure du magnétophone.

Après quelques essais, vous ne devriez plus avoir de problèmes.

Si vous voulez changer de programme, vous avez une alternative: A) appuyez sur la touche: RUN/STOP; si rien ne se produit, en la

maintenant toujours enfoncée, tapez une ou plusieurs fois sur la touche "RESTORE": l'écran devrait se vider et redevenir bleu avec le bord bleu clair. Alors écrivez "LOAD" et suivez les instructions habituelles. B) si vous n'arrivez vraiment pas à vous en sortir, n'ayez crainte, éteignez l'ordinateur et réallumez-le, et recommencez en suivant les instructions habituelles pour charger les programmes.

GESTION MAGASIN

Avec ce programme, on peut obtenir une gestion simple des stocks de magasin. Une fois donné le "RUN", l'ordinateur rappelle que, pour terminer chaque procédure, il suffit d'appuyer sur (*) et de taper "RETURN", il demande donc que l'on appuie sur une touche pour visualiser la liste générale. La voici, en détail (rappelons-nous que, pour obtenir la fonction désirée, il suffit d'appuyer sur le chiffre correspondant): "1 CHARGEMENT A PARTIR D'UN MAGNETOPHONE", on utilise cet ordre lorsque l'on a déjà constitué des archives sur une autre cassette et si l'on désire mettre les données à jour; si on se trouve dans cette situation, ré-enrouler la cassette, taper "1", appuyer sur une touche et enfoncer "PLAY", sur le magnétophone, puis attendre quelques secondes jusqu'à ce que la liste générale ne réapparaisse sur l'écran. Vérifiez immédiatement vos stocks en appuyant sur la touche 5.

"2 INTRODUCTION ARTICLES": cet ordre est utilisé lorsque l'on se sert du programme pour la première fois et que l'on doit créer les archives d'articles ou lorsque l'on doit ajouter en magasin un nouvel article qui n'avait jamais été possédé précédemment. L'ordinateur demande la description de l'article en question (ex. pointe-bic, cahiers, etc.); on appuie sur "RETURN" et on introduit la quantité possédée (suivie une fois encore de "RETURN"). On continue ainsi jusqu'à la fin des articles en dotation; à ce point, on écrit * (l'astérisque) et on appuie sur "RETURN" pour le retour au stock. Il est possible que certaines unités des articles enregistrés soient achetées ou vendues: dans le premier cas, en appuyant sur la touche "3", on effectue le "chargement"; on écrit l'article en question (ex. Bic), on tape "RETURN", puis on écrit le nombre des objets achetés, qui sera ajouté au nombre des articles enregistré précédemment. En cas de vente, on effectue le "DEGAGEMENT": la procédure est identique à la précédente, mais le nombre des articles sera soustrait à celui des articles déjà possédés. Pour vérifier les articles enregistrés et leur nombre, il suffira d'appuyer sur "5" et d'enfoncer une touche pour revenir au stock. Une fois l'introduction terminée ou la mise à jour des données, il est possible de conserver sur bande la situation de notre magasin: il suffira d'appuyer sur la touche "6" (fin travail) et d'introduire une cassette vierge ré-enroulée avant d'appuyer sur les touches "RECORD & PLAY" sur le magnétophone.

QUOTIENT INTELLECTUEL

Il s'agit d'un test scientifique efficace, comparable aux systèmes de mesure de l'intelligence mis au point par les psychologues STANDFORD-BINET, KUHLMANN, TERMAN, BINET et WECHSLER, qui permet d'établir, sur la base de paramètres conventionnels, le quotient intellectuel de la personne qui se soumet au feu nourri des quatre-vingt-dix questions posées par l'ordinateur. La durée maximum du test est de 45 minutes et il est donc conseillé de ne pas s'attarder sur les questions à problèmes.

Plutôt que de trouver une bonne réponse au bout de 10 minutes et ne pas avoir le temps de répondre à toutes les questions, il vaut mieux passer outre en appuyant sur la touche "SPACE" puis sur "RETURN". Le test est valable pour tous les âges à partir de douze ans.

CHASSE AU TRESOR

A bord de sa jeep, le joueur se trouve dans un champ miné qui cache à la fois de dangereux engins explosifs et de fabuleux trésors. Bien entendu, il doit s'efforcer de mettre la main sur les précieux coffres enterrés sans sauter en l'air et, sitôt sa tâche accomplie, il doit courir se mettre à l'abri dans le refuge.

C'est au joueur lui-même de décider combien de mines et combien de trésors son terrain renfermera.

Avant de partir, il consultera la carte qui lui indiquera l'emplacement des mines et des trésors. Une fois lancé, il lui faudra surveiller son niveau d'essence car il doit arriver au refuge avant que son réservoir ne soit à sec.

S'il le désire, le joueur peut consulter à nouveau la carte mais pendant ce temps il perdra du carburant!

A vous donc de jouer. Bonne chance et attention à l'essence!

ENFER 3D

"Perdez toute espérance ô vous qui entrez..."

Comme dans la Divine Comédie de Dante Alighieri, vous vous trouvez dans un enfer plein de surprises et, surtout, d'où il est très difficile de sortir.

Juste avant d'entrer en enfer 3D vous verrez le plan du labyrinthe: une petite flèche indiquera l'endroit où vous êtes et la sortie du labyrinthe vous apparaîtra également.

Il vous faudra avoir une très bonne mémoire et vous rappeler le parcours pour atteindre la sortie. Voici les instructions:
I pour avancer, J pour tourner à gauche, K pour tourner à droite, M pour aller en arrière, A pour abandonner et vous avouer vaincus. Attention, en marchant vous laisserez des traces, et donc s'il vous arrivait de revenir sur vos pas vous trouverez des boules plus grosses sur votre parcours. Amusez- vous bien et...

Bonne chance!...



SOFTHEQUE N. 1

MENSUEL - NOVEMBRE 1984

Directeur: Franco Bozzesi

Ont collaboré:
MICHELLE BLEIN
GEORGES RIEBEN
MAX CELLINI
ROBERTO TREPPIEDI
DIDIER DUCHESNE
ALDO CAMPANOZZI
GEORGETTE LHOPITAL
ALEX VALLONE
HELENE RACCAH
ALBERTO BARBATI
YOLANDE TERISSE
ANTONIO LUCARELLA
CATHERINE JUERY
DANIELE RIEFOLI
JEAN CAPOBIANCO

SOFTHEQUE est une création de PROMOPUBLICATIONS, S.a.r.l. au capital de 20.000 F. 312179195 B.R.C. Paris.
Rédaction, administration, vente, publicité, siège social: 34, Champs-Elysées, 75008 Paris. Tel. (1) 5634850
Distribué en France par: N.M.P.P.
Imprimerie: TIPO LITO FE.ZA. 17, rue Oslavia Milan - Italie
Directeur de la publication: Franco Bozzesi
Numéro de commission paritaire: en cours.
Depôt legal n. 50579 du 8 Juin 1984.


SOFTHEQUE
ORDINATEUR
1
Video-jeux et
programmes pour
ZX SPECTRUM
et VIC 20
ZX SPECTRUM:
• U.F.O. (16 K)
• WARGAME (48 K)
• ANATOMIE (48 K)
• GESTION MAGASIN (48 K)
VIC 20:
• QUOTIENT INTELLECTUEL
• CHASSE AU TRESOR
• ENFER 3D
• GESTION MAGASIN
Promopublications